INRIA

INSTITUT NATIONAL DE RECHERCHE EN INFORMATIQUE ET EN AUTOMATIQUE

Chinese Numbers, MIX, Scrambling, and Range Concatenation Grammars

Pierre Boullier

N ° 3614

Janvier 1999

THÈME 3

ISSN 0249-6399

Chinese Numbers, MIX, Scrambling, and Range Concatenation Grammars

Pierre Boullier*

Thème 3 — Interaction homme-machine,
images, données, connaissances
Projet Atoll

Rapport de recherche n°3614 — Janvier 1999 — 14 pages

Abstract: The notion of mild context-sensitivity was formulated in an attempt to express the formal power which is both necessary and sufficient to define the syntax of natural languages. However, some linguistic phenomena such as Chinese numbers and German word scrambling lie beyond the realm of mildly context-sensitive formalisms. On the other hand, the class of range concatenation grammars provides added power w.r.t. mildly context-sensitive grammars while keeping a polynomial parse time behaviour. In this report, we show that this increased power can be used to define the above-mentioned linguistic phenomena with a polynomial parse time of a very low degree.

Key-words: range concatenation grammars, mildly context-sensitive grammars, parsing time complexity, Chinese numbers, German scrambling, MIX.

(Résumé : tsvp)

* E-mail: Pierre.Boullier@inria.fr

Unité de recherche INRIA Rocquencourt
Domaine de Voluceau, Rocquencourt, BP 105, 78153 LE CHESNAY Cedex (France)
Téléphone : 01 39 63 55 11 - International : +33 1 39 63 55 11
Télécopie : (33) 01 39 63 53 30 - International : +33 1 39 63 53 30

Nombres chinois, MIX, constructions permutantes et grammaires à concaténation d'intervalles

Résumé : La notion de grammaire faiblement contextuelle résulte d'une tentative d'exprimer le pouvoir formel qui est à la fois nécessaire et suffisant pour définir la syntaxe des langues naturelles. Cependant, des phénomènes linguistiques comme les nombres en chinois ou les constructions permutantes de l'allemand échappent à de tels formalismes. D'un autre côté, la classe des grammaires à concaténation d'intervalles est un formalisme syntaxique qui est à la fois puissant et efficace ; bien que ses textes source puissent être aussi analysés en temps polynômial, ce formalisme est plus puissant que les grammaires faiblement contextuelles. Dans ce rapport, nous montrons que ces possibilités supplémentaires peuvent être utilisées pour définir les phénomènes linguistiques précédents avec un temps d'analyse polynômial à très faible degré.

Mots-clé : grammaires à concaténation d'intervalles, grammaires faiblement contextuelles, complexité en temps d'analyse, nombres en chinois, constructions permutantes en allemand, MIX.

1 Motivation

The notion of mild context-sensitivity originates in an attempt by [Joshi 85] to express the formal power needed to define natural languages (NLs). We know that context-free grammars (CFGs) are not adequate to define NLs since some phenomena are beyond their power (see [Shieber 85]). Popular incarnations of mildly context-sensitive (MCS) formalisms are tree adjoining grammars (TAGs) [Vijay-Shanker 87] and linear context-free rewriting (LCFR) systems [Vijay-Shanker, Weir, and Joshi 87]. However, there are some linguistic phenomena which are known to lie beyond MCS formalisms. Chinese numbers have been studied in [Radzinski 91] where it is shown that the set of these numbers is not a LCFR language and that it appears also not to be MCS. Scrambling is a word-order phenomenon which also lies beyond LCFR systems (see [Becker, Rambow and Niv 92]).

On the other hand, in [Boullier 98a], we presented range concatenation grammars (RCGs), a syntactic formalism which is a variant of literal movement grammars (LMGs), described in [Groenink 97], and which is also related to the framework of LFP developed by [Rounds 88]. This formalism extends CFGs and is even more powerful than LCFR systems, while staying computationally tractable: its sentences can be parsed in polynomial time.

The purpose of this paper is to study whether the extra power of RCGs (over LCFR systems) is sufficient to deal with Chinese numbers and German scrambling phenomena.

2 Range Concatenation Grammars

This section introduces the notion of RCG in a rather informal way and presents some of its properties. More details appear in [Boullier 98a] and [Boullier 99].

Definition 1 *A positive range concatenation grammar (PRCG) $G = (N, T, V, P, S)$ is a 5-tuple where N is a finite set of predicate names, T and V are finite, disjoint sets of terminal symbols and variable symbols respectively, $S \in N$ is the start predicate name, and P is a finite set of clauses*

$$\psi_0 \rightarrow \psi_1 \dots \psi_m$$

where $m \geq 0$ and each of $\psi_0, \psi_1, \dots, \psi_m$ is a predicate of the form

$$A(\alpha_1, \dots, \alpha_p)$$

where $p \geq 1$ is its arity, $A \in N$ and each of $\alpha_i \in (T \cup V)^*$, $1 \leq i \leq p$, is an argument.

Each occurrence of a predicate in the RHS of a clause is a predicate *call*, it is a predicate *definition* if it occurs in its LHS. Clauses which define predicate A are called A-clauses. This definition assigns a fixed arity to each predicate name. The arity of the start predicate is one. The *arity* k of a grammar (we have a k-PRCG), is the maximum arity of its predicates.

Early occurring lower case letters such as $a, b, c, \dots$ will denote terminal symbols, while late occurring upper case letters such as X, Y, Z will denote elements of V.

The language defined by a PRCG is based on the notion of *range*. For a given input string $w = a_1 \dots a_n$ a range is a couple (i, j), $0 \leq i \leq j \leq n$ of integers which denotes the occurrence of some substring $a_{i+1} \dots a_j$ in w. The number $j - i$ is its *size*. We will use several equivalent denotations for ranges: an explicit dotted notation like $w_1 \bullet w_2 \bullet w_3$ or, if w_2 extends from positions $i + 1$ through j, a tuple notation $\langle i..j \rangle_w$, or $\langle i..j \rangle$ when w is understood or of no importance. For a range $\langle i..j \rangle$, i is its *lower bound* and j is its *upper bound*. If $i = j$, we have an *empty* range. Of course, only consecutive ranges can be concatenated into new ranges. In any PRCG, terminals, variables and arguments in a clause are supposed to be bound to ranges by a substitution mechanism. An *instantiated clause* is a clause in which variables and arguments are consistently (w.r.t. the concatenation operation) replaced by ranges; its components are *instantiated predicates*.

For example, $A(\langle g..h \rangle, \langle i..j \rangle, \langle k..l \rangle) \rightarrow B(\langle g{+}1..h \rangle, \langle i{+}1..j{-}1 \rangle, \langle k..l{-}1 \rangle)$ is an instantiation of the clause $A(aX, bYc, Zd) \rightarrow B(X, Y, Z)$ if the source text $a_1 \dots a_n$ is such that $a_{g+1} = a, a_{i+1} = b, a_j = c$ and $a_l = d$. In this case, the variables X, Y and Z are bound to $\langle g{+}1..h \rangle$, $\langle i{+}1..j{-}1 \rangle$ and $\langle k..l{-}1 \rangle$ respectively.[1]

A *derive* relation, denoted by $\underset{G,w}{\Rightarrow}$, is defined on strings of instantiated predicates. If an instantiated predicate is the LHS of some instantiated clause, it can be replaced by the RHS of that instantiated clause.

Definition 2 *The language of a PRCG $G = (N, T, V, P, S)$ is the set*

$$\mathcal{L}(G) \quad = \quad \{w \mid S(\bullet w \bullet) \overset{+}{\underset{G,w}{\Rightarrow}} \varepsilon\}$$

An input string $w = a_1 \dots a_n$ is a sentence iff the empty string (of instantiated predicates) can be derived from $S(\langle 0..n \rangle)$.

The arguments of a given predicate may denote discontinuous or even overlapping ranges. Fundamentally, a predicate name A defines a notion (property, structure, dependency, ...) between its arguments, whose ranges can be arbitrarily scattered over the source text. What is "between" its arguments is *not* the responsibility of A, and is described (if at all) somewhere else. PRCGs are therefore well suited to describe long distance dependencies. Overlapping ranges arise as a consequence of the non-linearity of the formalism. For example, the same variable (denoting the same range) may occur in different arguments in the RHS of some clause, expressing different views (properties) of the same portion of the source text.

Note that the order of RHS predicates in a clause is of no importance.

As an example of a PRCG, the following set of clauses describes the three-copy language $\{www \mid w \in \{a, b\}^*\}$ which is not a CFL and even lies beyond the formal power of TAGs.

$$\begin{array}{lcl} S(XYZ) & \rightarrow & A(X, Y, Z) \\ A(aX, aY, aZ) & \rightarrow & A(X, Y, Z) \\ A(bX, bY, bZ) & \rightarrow & A(X, Y, Z) \\ A(\varepsilon, \varepsilon, \varepsilon) & \rightarrow & \varepsilon \end{array}$$

[1] Often, instead of saying *the range which is bound to X or denoted by X*, we will say, the range X, or even instead of *the string whose occurrence is denoted by the range which is bound to X*, we will say the string X.

Definition 3 *A negative range concatenation grammar (NRCG) $G = (N, T, V, P, S)$ is a 5-tuple, like a PRCG, except that some predicates occurring in RHS, have the form $\overline{A(\alpha_1, \ldots, \alpha_p)}$, $A \in N$.*

A predicate call of the form $\overline{A(\alpha_1, \ldots, \alpha_p)}$ is said to be a *negative predicate call.* The intuitive meaning is that a negative predicate succeeds iff its positive counterpart (always) fails. Therefore this definition is based on a "negation by failure" rule. However, in order to avoid inconsistencies occurring when an instantiated predicate is defined in terms of its negative counterpart, we prohibit derivations exhibiting this possibility. Thus we only allow so called *consistent* derivations. We say that a grammar is consistent if all of its derivations are consistent.

Definition 4 *A range concatenation grammar (RCG) is a PRCG or a NRCG.*

The PRCG (resp. NRCG) term will be used to underline the absence (resp. presence) of negative predicates.

In [Boullier 98a], we presented a parsing algorithm which, for an RCG G and an input string of length n, produces a parse forest in time polynomial with n and linear with $|G|$. The degree of this polynomial is at most the number of free bounds in any clause.

Intuitively, if we consider the instantiation of a clause, all its terminal symbols, variable, arguments are bound to ranges. This means that each position in its arguments is mapped onto a *source index*, a position in the source text. However, the knowledge of all couples (argument position, source index) is usually not necessary to fully defined the mapping. For example, if XaY is some argument, if $X \bullet aY$ denotes a position in this argument, and if $(X \bullet aY, i)$ is an element of the mapping, we know that $(Xa \bullet Y, i+1)$ must be another element. Argument positions $p_1, \ldots, p_j$ for which there is a mapping m s.t. the source index $m(p_i)$ of some p_i cannot be deduced from the images of the other argument positions, are called *free bounds*. Of course, number of free bounds, means *minimum* number of argument positions needed to deduce all the other positions in the clause.

2.1 Predefined Predicates

The current implementation has a certain number of predefined predicate names among which we quote here *len*, *eqlen* or *eq*:

$len(l, X)$**:** checks that the size of the range denoted by the variable X is the integer l.

$eqlen(X, Y)$**:** checks that the size of X and Y are equal.

$eq(X, Y)$**:** checks that the substrings selected by the ranges X and Y are equal.

It must be noted that these predefined predicates do not increase the formal power of RCGs since each of them can be defined by a pure RCG. For example, the predicate $len(1, X)$ can be defined by the clause $len_1(t) \rightarrow \varepsilon$ which is a schema over all terminals $t \in T$. Their

introduction is not only justified by the fact that they are more efficiently implemented than their RCG defined counterpart but mainly because they convey some static information which can be used to decrease the number of free bounds and thus lead to an improved parse time.

For example, the predicate call $len(3, X)$, indicates that the size of X is 3, and this means that the positions in the arguments before and after any occurrence of the variable X are not both free. In an analogous manner, $eq(X, Y)$ ($eq \Longrightarrow eqlen$), implies that the four lower and upper bounds of X and Y are not all free.

3 Chinese Numbers & RCGs

The number-name system of Chinese, specifically the Mandarin dialect, allows large number names to be constructed in the following way. The name for 10^{12} is *zhao* and the word for five is *wu*. The sequence *wu zhao zhao wu zhao* is a welled-formed Chinese number name (i.e. $5\ 10^{24} + 5\ 10^{12}$) although *wu zhao wu zhao zhao* is not: the number of consecutive *zhao*'s must strictly decrease from left to right. All the well-formed number names composed only of instances of *wu* and *zhao* form the set

$$\{wu\ zhao^{k_1}\, wu\ zhao^{k_2} \ldots wu\ zhao^{k_p} \mid k_1 > k_2 > \ldots > k_p > 0\}$$

which can be abstracted as

$$\text{CN} \quad = \quad \{ab^{k_1} ab^{k_2} \ldots ab^{k_p} \mid k_1 > k_2 > \ldots > k_p > 0\}$$

These numbers have been studied in [Radzinski 91], where it is shown that CN is not a LCFR language but an Indexed Language (IL) [Aho 68]. Radzinski also argued that CN also appears not to be MCS and moreover he says that he fails "*to find a well-studied and attractive formalism that would seem to generate Numeric Chinese without generating the entire class of ILs (or some non-ILs)*". In [Boullier 98a], [Boullier 98b] and [Boullier 99] we have shown that RCGs are not MCS and are strictly more powerful than LCFR systems.

We will show that the language of the following RCG is CN.

$$\begin{array}{lcl}
S(aX) & \rightarrow & A(X, aX, X) \\
A(W, TX, bY) & \rightarrow & len(1, T)\ A(W, X, Y) \\
A(WaY, X, aY) & \rightarrow & \overline{len(0, X)}\ A(Y, W, Y) \\
A(W, X, \varepsilon) & \rightarrow & \overline{len(0, X)}\ \overline{len(0, W)}
\end{array}$$

Intuitively, the idea of this grammar is the following. Its core is the predicate A of arity three. For the time being, let's forget about its first argument. Let's call b^{k_i} the i^{th} slice. The string denoted by the third argument has always the form $b^{k_i - l} ab^{k_{i+1}} \ldots$, it is a suffix of the source text, its prefix $ab^{k_1} \ldots ab^{k_i - 1} ab^{l}$ has already been examined. The property of the range of the second argument is to have a size which is strictly greater than $k_i - l$, the number of b's of the current slice still to be processed. The leading b's of the third

argument and the leading symbols of the second argument are simultaneously scanned (and skipped) by the second clause, until either the next slice is introduced (by an a) in the third clause, or the whole source text is exhausted in the fourth clause. When the processing of a slice is completed, we must check that the size of the second argument is not null (i.e. that $k_{i-1} > k_i$). This is performed by the calls $\overline{len(0, X)}$ in the third and fourth clause. However, doing that, the i^{th} slice has been erased, but, in order for the process to continue, this slice must be "rebuilt" since it will be used in the second argument to process the next slice. This reconstruction process is performed with the help of the first argument. At the beginning of the processing of a new slice, say the i^{th}, both the first and third argument denote the same string $b^{k_i}ab^{k_{i+1}}\ldots$. The first argument will stay unchanged while the leading b's of the third argument are processed (see the second clause). When the processing of the i^{th} slice is completed, and if it is not the last one (case of the third clause), the first and third argument respectively denote the strings $b^{k_i}ab^{k_{i+1}}\ldots$ and $ab^{k_{i+1}}\ldots$. Thus, the i^{th} slice b^{k_i} can be extracted "by difference", it is the string W if the first and third argument are respectively WaY and aY (see the third clause). Last, the whole process is initialized by the first clause. The first and third argument of A are equal, since we start a new slice, the second argument is forced to be strictly greater than the third, doing that, we are sure that it is strictly greater than k_1, the size of the first slice. Remark that the test $\overline{len(0, W)}$ in the fourth clause checks that the size k_p of the rightmost slice is not null, as stipulated in the language formal definition.

If we look at this grammar, for any input string of length n, we can easily see that the maximum number of predicate calls is $n + 1$, this number is an upper limit which is only reached for (acceptable) sentences. Thus, the RCG defining the language CN has a linear parse time behavior.

We have therefore shown that Chinese numbers can be parsed in linear time by an RCG.

4 MIX & RCGs

MIX is a language that consists of strings in $\{a, b, c\}^*$ such that each string contains the same number of occurrences of each letter. This language was originally described by Emmon Bach. MIX is interesting because it has a very simple and intuitive characterization. However, Gazdar reported[2] that MIX may well be outside the class of ILs (as conjectured by Bill Marsh in an unpublished 1985 ASL paper). It has turned out to be a very difficult problem. In [Joshi, Vijay-Shanker, and Weir 91] the authors have shown that MIX can be defined by a variant of TAGs with local dominance and linear precedence (TAG(LD/LP)), but very little is known about that class of grammars, except that, as TAGs, they continue to satisfy the constant growth property. We will see in the next section how MIX can be related with scrambling phenomena. Below, we will show that MIX is an RCL which can be recognized in linear time.

Consider the RCG

[2] See http://www.ccl.kuleuven.ac.be/LKR/dtr/mix1.dtr.

$$
\begin{array}{lll}
1: & S(X) & \rightarrow \quad M(X,X,X) \\
2: & M(aX,bY,cZ) & \rightarrow \quad M(X,Y,Z) \\
3: & M(TX,Y,Z) & \rightarrow \quad len(1,T)\ \overline{a(T)}\ M(X,Y,Z) \\
4: & M(X,TY,Z) & \rightarrow \quad len(1,T)\ \overline{b(T)}\ M(X,Y,Z) \\
5: & M(X,Y,TZ) & \rightarrow \quad len(1,T)\ \overline{c(T)}\ M(X,Y,Z) \\
6: & M(\varepsilon,\varepsilon,\varepsilon) & \rightarrow \quad \varepsilon \\
7: & a(a) & \rightarrow \quad \varepsilon \\
8: & b(b) & \rightarrow \quad \varepsilon \\
9: & c(c) & \rightarrow \quad \varepsilon
\end{array}
$$

The simple idea behind this grammar is the following. The source text is concurrently scanned three times by the three arguments of the predicate M (see the predicate call $M(X,X,X)$ in the first clause). The first, second and third argument of M respectively only deal with the letters a, b and c. If the leading letter of any argument (which at any time is a suffix of the source text) is not the right letter, this letter is skipped. The third clause only process the first argument of M (the two others are passed unchanged), in skipping any letter which is not an a. The analogous holds for the fourth and fifth clauses which respectively only consider the second and third argument of M, looking for a leading b or c. Note that the knowledge that a letter is not the right one is acquired via a negative predicate call because this allows for an easy generalization to any number of letters. In the case where the three leading letters are respectively a, b and c, they are simultaneously skipped (see clause #2) and the clause #6 is eventually instantiated if and only if the input string contains the same number of occurrences of each letter.

It is not difficult to see that each letter is examined at most (if the input word is a sentence) three times (one for each argument) and thus the parse time complexity of this grammar is linear.

Of course, we can think of several generalizations of the MIX language. Below, we will examine two of them, one in which the relation between the number of occurrences of each letter is not the equality and a second in which the number of letters is not limited to three.

As a first generalization of MIX, consider the language

$$\mathrm{MIX}_{213} \quad = \quad \{w \mid w = \sigma(a^{2k}b^kc^{3k}) \text{ and } \sigma \text{ is a permutation}\}.$$

From the previous grammar, we simply have to change the second clause into

$$2': \quad M(aX,bY,cZ) \quad \rightarrow \quad M_{111}(X,Y,Z)$$

and to add the following clauses

$$
\begin{array}{lcl}
M_{111}(TX,Y,Z) & \rightarrow & len(1,T)\ \overline{a(T)}\ M_{111}(X,Y,Z) \\
M_{111}(X,Y,TZ) & \rightarrow & len(1,T)\ \overline{c(T)}\ M_{111}(X,Y,Z) \\
M_{111}(aX,Y,cZ) & \rightarrow & M_{212}(X,Y,Z) \\
M_{212}(X,Y,TZ) & \rightarrow & len(1,T)\ \overline{c(T)}\ M_{212}(X,Y,Z) \\
M_{212}(X,Y,cZ) & \rightarrow & M(X,Y,Z)
\end{array}
$$

where the predicate name M_{ijk} simply means that the number of a's, b's and c's already seen in the current group is respectively i, j and k. Languages like $\{a^p b^q c^r / w \mid w = \sigma(a^{pk} b^{qk} c^{rk})\}$ where the prefix, before the slash character, of each sentence (dynamically) specifies the relative quantities of a, b and c's are also RCLs.

If we generalize the MIX language to any number of letters, we still get an RCL. Of course the language in which the three letters a, b and c are replaced by any number p of different letters can be processed by an RCG in which the arity of the M predicate is p, each argument being devoted to a particular letter.

However, we can also think of a generalization which consists of strings in R^*, where R is any subset of the terminal vocabulary T, such that each word contains the same number of occurrences of each letter of R. In such a case the arities of all possible M predicates range from 1 to $|T|$ and each such predicate of arity p should examine all possible combinations of p letters among $|T|$ possibilities.

Fortunately, we can think of another way of handling this problem. Each letter can be examined against a reference letter by a M predicate with two arguments, one holding a suffix of the source text in which the occurrences of the reference letter are counted and a second in which the occurrences of the second letter are counted. Moreover, to be fully general, we can add two arguments holding on the one hand the reference letter itself and on the other hand the current letter. Thus the predicate call $M_4(T,X,T_1,Y)$, where the variables T and T_1 denote (occurrences) of two letters, say t and t_1, is true if and only if the strings denoted by the variables X and Y respectively contain the same number of occurrences of t and t_1.[3]

$$
\begin{array}{lcl}
S(\varepsilon) & \rightarrow & \varepsilon \\
S(TX) & \rightarrow & len(1,T)\ A(T,TX,TX) \\
A(T,W,T_1X) & \rightarrow & len(1,T_1)\ M_4(T,W,T_1,W)\ A(T,W,X) \\
A(T,W,\varepsilon) & \rightarrow & \varepsilon \\
M_4(T,T'X,T_1,T_1'Y) & \rightarrow & eq(T,T')\ eq(T_1,T_1')\ M_4(T,X,T_1,Y) \\
M_4(T,T'X,T_1,Y) & \rightarrow & len(1,T')\ \overline{eq(T,T')}\ M_4(T,X,T_1,Y) \\
M_4(T,X,T_1,T_1'Y) & \rightarrow & len(1,T_1')\ \overline{eq(T_1,T_1')}\ M_4(T,X,T_1,Y) \\
M_4(T,\varepsilon,T_1,\varepsilon) & \rightarrow & \varepsilon
\end{array}
$$

[3]Note that, throughout all calls to M_4, the reference letter T will be constant.

In this grammar[4], the reference terminal is the leading input letter, if any (see the first and second clauses). The A-clauses strip the input string from left-to-right until completion (see the last argument) and are in charge of calling $M_4(T, W, T_1, W)$, where T denotes the reference letter, T_1 the letter to be examined, and W the whole source text. We can easily see that the time complexity behavior of this grammar is quadratic in the length n of the input string. For each couple (T, T_1), the call $M_4(T, W, T_1, W)$ gives its answer in time linear in $|W|$, and the number of such predicate calls is also linear in n.

Of course, many improvements could be thought of, but this is not the right place for their study.

5 Scrambling & RCGs

Scrambling is a word-order phenomenon which occurs in several languages such as German, Japanese, Hindi, ... and which is known to be beyond the formal power of TAGs (see [Becker, Joshi and Rambow 91]). In [Becker, Rambow and Niv 92], the authors even show that LCFR systems cannot derive scrambling. This is of course also true for multi-components TAGs (MCTAGs) (see [Rambow 94]). In [Groenink 97], p. 171, the author said that "*simple LMG formalism does not seem to provide any method that can be immediately recognized as solving such problems*". We will show below that scrambling can be expressed within the RCG framework.

Scrambling can be seen as a leftward movement of arguments (nominal, prepositional or clausal). Groenink notices that similar phenomena also occur in Dutch verb clusters, where the order of verbs (as opposed to objects) can in some case be reversed.

In [Becker, Rambow and Niv 92], from the following German example

...daß	[dem Kunden]$_i$	[den Kuehlschrank]$_j$	bisher noch	niemand
...that	the client (DAT)	the refrigerator (ACC)	so far yet	no-one (NOM)

t_i [[t_j	zu reparieren]	zu versuchen]	versprochen	hat.
	to repair	to try	promised	has.

...that so far no-one has promised the client to try to repair the refrigerator.

the authors argued that scrambling may be "doubly unbounded" in the sense that:

- there is no bound on the distance over which each element can scramble;
- there is no bound on the number of unbounded dependencies that can occur in one sentence.

[4]Note that this grammar accepts both ε and t^+, $t \in T$ as sentences since the subset R can be either empty or a singleton.

They used the language

$$\text{SCR} \quad = \quad \{\sigma(n_0, \ldots, n_m)\ v_0 \ldots v_m \mid m \geq 0 \text{ and } \sigma \text{ is a permutation}\}$$

as a formal representation for a subset of scrambled German sentences, where it is assumed that each verb v_i has exactly one overt nominal argument n_i.

The following RCG defines a generalization of SCR in which on the one hand a verb v_i and its nominal argument n_i, can be freely mixed (the argument do not necessarily precede its corresponding verb) and which, on the other hand, allows several occurrences of the pair (n_i, v_i). We assume that a terminal t is a verb if and only if $v(t)$ is true, and that h is the injective mapping between a verb and its argument. We have

$$\begin{array}{lcl}
S(W) & \rightarrow & A(W,W) \\
A(W,TX) & \rightarrow & len(1,T)\ v(T)\ B(T,W,W)\ A(W,X) \\
A(W,\varepsilon) & \rightarrow & \varepsilon \\
B(T,T_1X,W) & \rightarrow & len(1,T_1)\ \overline{eq(T,T_1)}\ B(T,X,W) \\
B(T,T_1X,W) & \rightarrow & eq(T,T_1)\ C(T,X,W) \\
C(T,X,T_1Y) & \rightarrow & len(1,T_1)\ \overline{h(T,T_1)}\ C(T,X,Y) \\
C(T,X,T_1Y) & \rightarrow & len(1,T_1)\ h(T,T_1)\ M_4(T,X,T_1,Y) \\
v(v_0) & \rightarrow & \varepsilon \\
\ldots & & \\
v(v_m) & \rightarrow & \varepsilon \\
h(v_0,n_0) & \rightarrow & \varepsilon \\
\ldots & & \\
h(v_m,n_m) & \rightarrow & \varepsilon
\end{array}$$

where the predicate name M_4 is defined as in the previous section. In this grammar, the A-clauses extract from left to right from the source text w all the occurrences of each verb v (see the last argument). For each such occurrence of v, the B predicate finds the first occurrence of v and remembers the corresponding suffix X of w and calls $C(T,X,W)$, where T is an occurrence of the verb v to process, X is the suffix of the source text following the first occurrence of v (from left to right), and W is the whole source text. The C-clauses find the first occurrence of the noun n associated with v, and remember the corresponding suffix Y of w. When M_4 is called, it checks that the number of occurrences of v in X is the number of occurrences of n in Y. We can easily see that,

for each occurrence of a verb v in w, the discovery of the leftmost occurrence of v, the leftmost occurrence of n and the processing of the corresponding M_4 call take a time linear in $n = |w|$. Since each occurrence of any verb is found during a single scan of w, the total parse time for this grammar is quadratic in n.

Note that the definition of what terminal is a verb, and the definition of the association of a verb to its nominal argument have respectively been performed via the predicates name v and h, because this allows for more modularity and makes generalizations easier.

If the group of nominal arguments must precede the group of verbs, we can use the closure of RCLs under intersection and add the clause

$$S'(W) \quad \rightarrow \quad n^*v^*(W)\ S(W)$$

where (the RCG version of) the regular language $\{n_0, \ldots, n_m\}^*\{v_0, \ldots, v_m\}^*$ is defined by the predicate name n^*v^*. The parse time is kept quadratic at worst.

However, in [Becker, Joshi and Rambow 91] we can find an example

...daß	[des Verbrechens]$_k$	[der Detektiv]$_i$	[den Verdächtigen]$_j$ dem Klienten
...that	the crime (GEN)	the detective (NOM)	the suspect (ACC) the client (DAT)

[PRO$_i$ t_j t_k	zu überführen]	versprochen	hat.
	to indict	promised	has.

...that the detective has promised the client to indict the suspect of the crime.

where the verb of the embedded clause sub-categorizes for three NPs, one of which is an empty subject (PRO).

In order to deal with this phenomenon, we can use a similar method to the one used in the previous section with the predicate names M_{ijk}. If a verb sub-categorizes for multiple NPs, the number of NPs already seen can be subtracted by using different (static) predicate names. This (dis)counting can even be performed dynamically with a single predicate name containing a supplementary argument whose size is the number of nouns still to be discovered.

6 Conclusion

In [Boullier 98a], [Boullier 98b] and [Boullier 99] we have shown that the class of RCGs is a syntactic formalism which seems very promising since it has many interesting properties among which we can quote its power, above that of LCFR systems; its efficiency, with polynomial time parsing; its modularity; and the fact that the output of its parsers can be viewed as shared parse forests. It can thus be used as is to define languages or it can be used as an intermediate (high-level) representation. This last possibility comes from the fact that many popular formalisms can be translated into equivalent RCGs, without loosing any efficiency. For example, TAGs can be translated into equivalent RCGs which can be parsed in $\mathcal{O}(n^6)$ time.

The formal power of RCGs can be used for example to define exotic toy languages such as $\{a^{2^k}\}$, $\{a^{k^2}\}$ or $\{w \mid |w| \text{ is a prime}\}$ with very efficient parse time. In this paper, we have shown that this extra formal power can be used in NL processing. We turn our attention to the two phenomena of Chinese numbers and German scrambling which are both beyond the formal power of MCS formalisms. To our knowledge, Chinese numbers were only known to be an IL and it was not even known whether the scrambling phenomenon can be described by an IG. We have seen that these phenomena can both be defined with RCGs. Moreover,

the corresponding parse time is polynomial with a very low degree. During this work we have also noted that scrambling can be related to the famous MIX language, which has been classified as a linear parse time RCL.

References

[Aho 68] Alfred V. Aho. 1968. Indexed grammars – an extension of context-free grammars. In *Journal of the ACM*, Vol. 15, pages 647–671.

[Becker, Joshi and Rambow 91] Tilman Becker, Aravind Joshi, Owen Rambow. 1991. Long distance scrambling and tree adjoining grammars. In *Proceedings of the fifth Conference of the European Chapter of the Association for Computational Linguistics (EACL'91)*, pages 21–26.

[Becker, Rambow and Niv 92] Tilman Becker, Owen Rambow and Michael Niv. 1992. The Derivational Generative Power of Formal Systems or Scrambling is Beyond LCFRS. In *Technical Report IRCS-92-38*, Institute for Research in Cognitive Science, University of Pennsylvania, Philadelphia, PA.

[Boullier 98a] Pierre Boullier. 1998. Proposal for a Natural Language Processing Syntactic Backbone. In *Research Report No 3342* at `http://www.inria.fr/RRRT/RR-3342.html`, INRIA-Rocquencourt, France, Jan. 1998, 41 pages.

[Boullier 98b] Pierre Boullier. 1998. A Generalization of Mildly Context-Sensitive Formalisms. In *Proceedings of the Fourth International Workshop on Tree Adjoining Grammars and Related Frameworks (TAG+4)*, University of Pennsylvania, Philadelphia, PA, 1–3 August, pages 17–20.

[Boullier 99] Pierre Boullier. 1998. A Cubic Time Extension of Context-Free Grammars. In *Research Report No 3611* at `http://www.inria.fr/RRRT/RR-3611.html`, INRIA-Rocquencourt, France, Jan. 1999, 28 pages.

[Groenink 97] Annius V. Groenink. 1997. SURFACE WITHOUT STRUCTURE Word order and tractability issues in natural language analysis. PhD thesis, Utrecht University, The Netherlands, Nov. 1977, 250 pages.

[Joshi 85] Aravind K. Joshi. 1985. How much context-sensitivity is necessary for characterizing structural descriptions — Tree Adjoining Grammars. In *Natural Language Processing — Theoritical, Computational and Psychological Perspective*, D. Dowty, L. Karttunen, and A. Zwicky, editors, Cambridge University Press, New-York, NY.

[Joshi, Vijay-Shanker, and Weir 91] Aravind K. Joshi, K. Vijay-Shanker, David Weir. 1991. The convergence of mildly context-sensitive grammatical formalisms. In *Foundational Issues in Natural Language Processing*, P. Sells, S. Shieber, and T. Wasow editors, MIT Press, Cambridge, Mass.

[Radzinski 91] Daniel Radzinski. 1991. Chinese Number-Names, Tree Adjoining Languages, and Mild Context-Sensitivity. In *Computational Linguistics*, 17(3), pages 277–299.

[Rambow 94] Owen Rambow. 1994. Formal and Computational Aspects of Natural Language Syntax. In *PhD Thesis*, University of Pennsylvania, Philadelphia, PA.

[Rounds 88] William C. Rounds. 1988. LFP: A Logic for Linguistic Descriptions and an Analysis of its Complexity. In *ACL Computational Linguistics*, Vol. 14, No. 4, pages 1–9.

[Shieber 85] Stuart M. Shieber. 1985. Evidence against the context-freeness of natural language. In *Linguistics and Philosophy*, Vol. 8, pages 333–343.

[Vijay-Shanker 87] K. Vijay-Shanker. 1987. A study of tree adjoining grammars. *PhD thesis*, University of Pennsylvania, Philadelphia, PA.

[Vijay-Shanker, Weir, and Joshi 87] K. Vijay-Shanker, David J. Weir, Aravind K. Joshi. 1987. Characterizing Structural Descriptions Produced by Various Grammatical Formalisms. In *Proceedings of the 25th Meeting of the Association for Computational Linguistics (ACL'87)*, Stanford University, CA, pages 104–111.

Unit´e de recherche INRIA Lorraine, Technopôle de Nancy-Brabois, Campus scientifique,
615 rue du Jardin Botanique, BP 101, 54600 VILLERS LÈS NANCY
Unit´e de recherche INRIA Rennes, Irisa, Campus universitaire de Beaulieu, 35042 RENNES Cedex
Unit´e de recherche INRIA Rhône-Alpes, 655, avenue de l'Europe, 38330 MONTBONNOT ST MARTIN
Unit´e de recherche INRIA Rocquencourt, Domaine de Voluceau, Rocquencourt, BP 105, 78153 LE CHESNAY Cedex
Unit´e de recherche INRIA Sophia-Antipolis, 2004 route des Lucioles, BP 93, 06902 SOPHIA-ANTIPOLIS Cedex

Éditeur
INRIA, Domaine de Voluceau, Rocquencourt, BP 105, 78153 LE CHESNAY Cedex (France)
http://www.inria.fr
ISSN 0249-6399